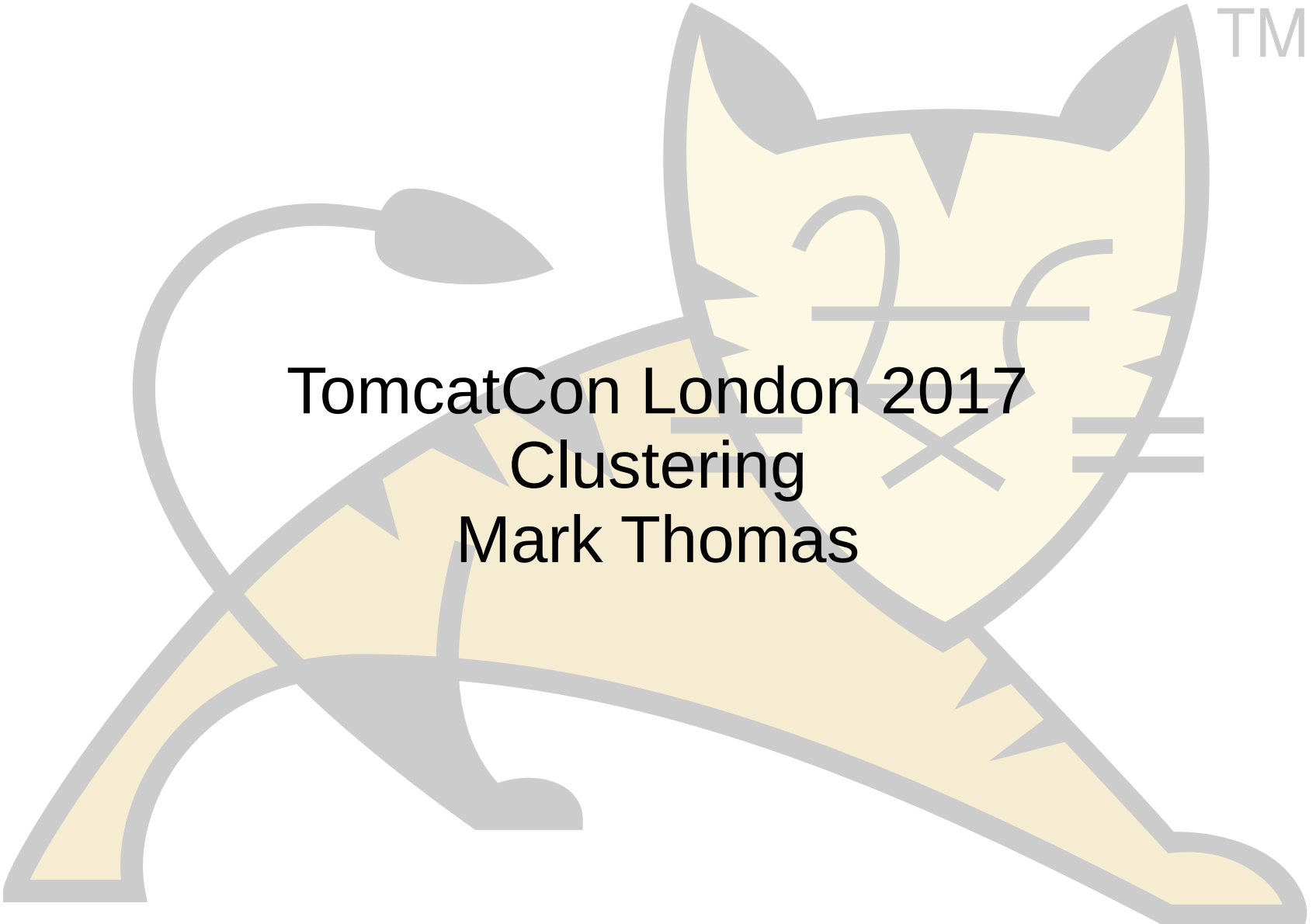
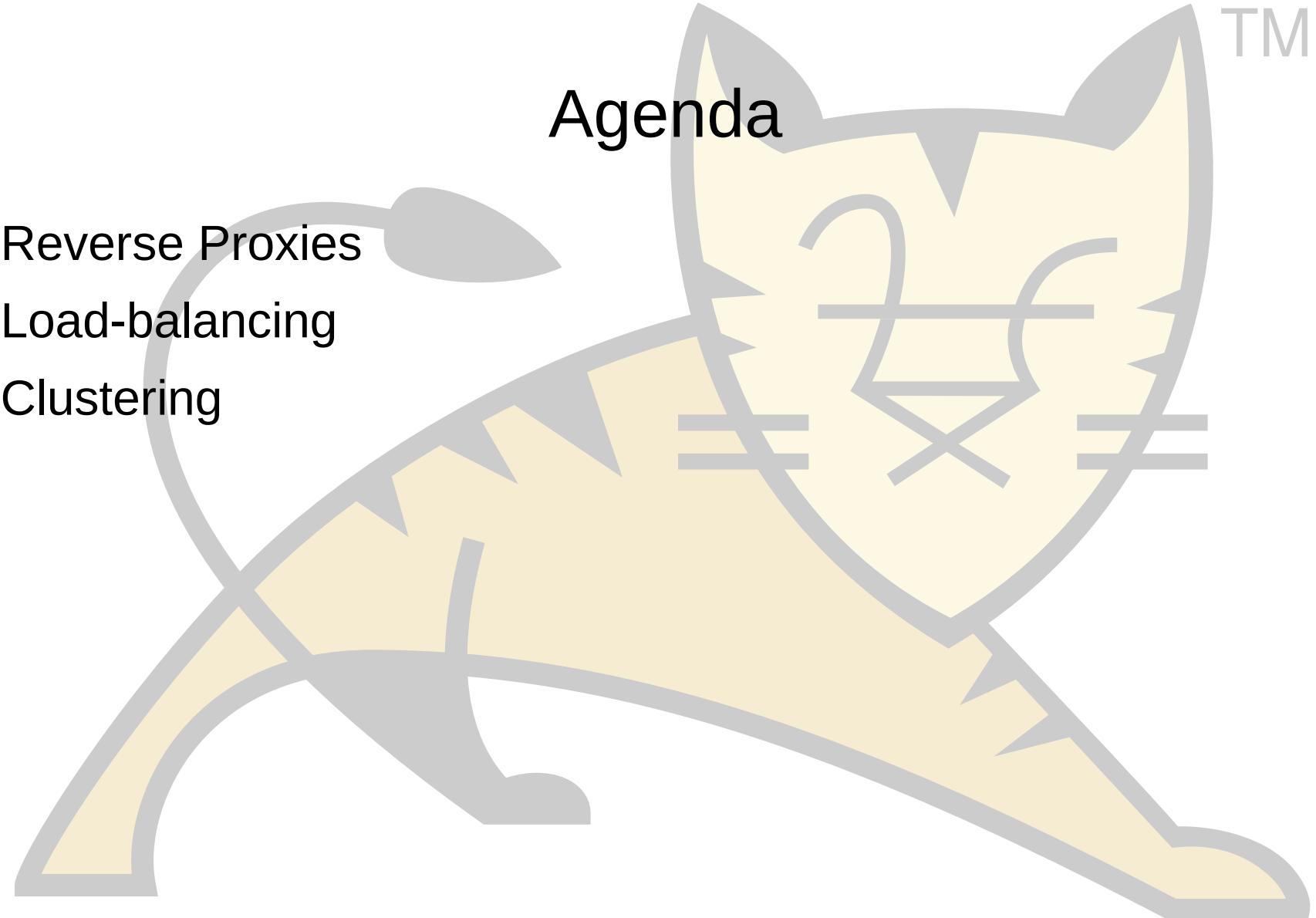


TomcatCon London 2017
Clustering
Mark Thomas

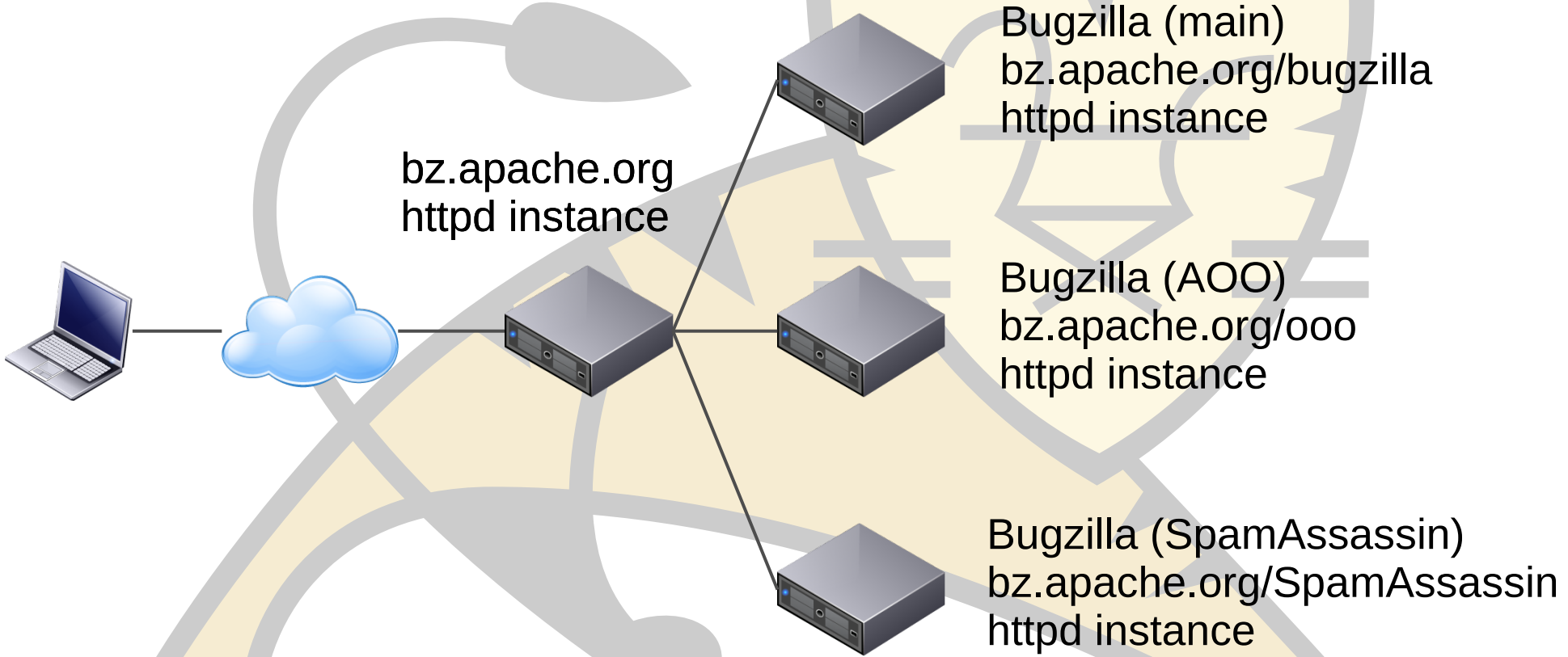
Agenda

- Reverse Proxies
- Load-balancing
- Clustering



Reverse Proxies

Reverse Proxy

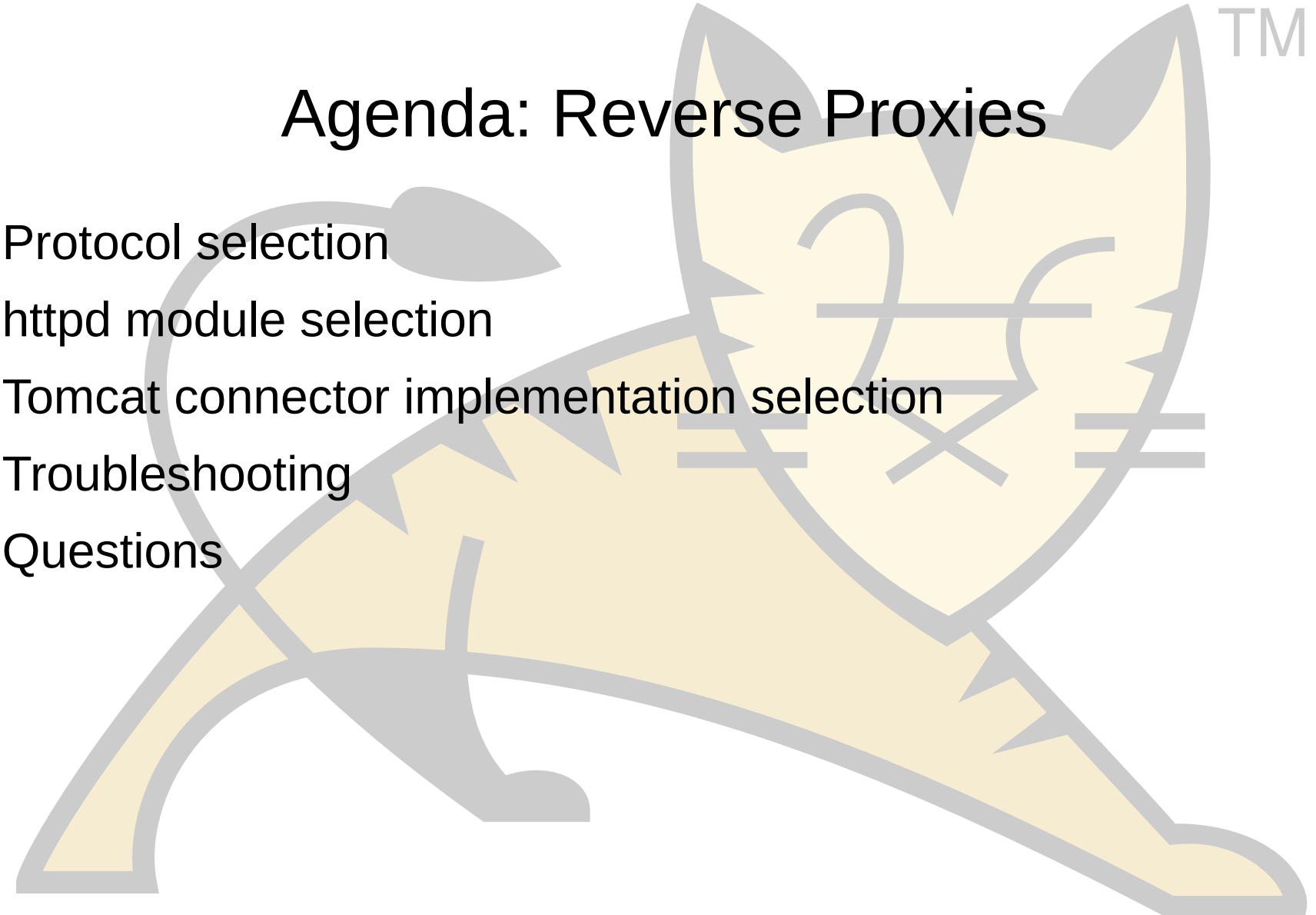


Reverse Proxy

- Looks like a single host to the clients
- Usually multiple hosts
- Different services on different hosts
  - May also be geographically distributed
- Can be used to add features
  - e.g. Use httpd as a reverse proxy for Tomcat to add Windows authentication (no longer necessary)

Agenda: Reverse Proxies

- Protocol selection
- httpd module selection
- Tomcat connector implementation selection
- Troubleshooting
- Questions



Protocol Selection

- Two options
  - AJP
  - HTTP
- Best choice depends on circumstances
  - No clear winner
- Both support persistent connections
  - On a fast LAN or the same machine this makes little difference

Protocol Selection: AJP

- Not a binary protocol
  - Common headers and values encoded
  - Other values in plain text
  - Request and response bodies in plain text
- Request headers must fit in a single AJP message
  - Default 8192
  - Max 65536

Protocol Selection: AJP

- Supports passing of TLS termination information
- Does not directly support encryption
  - IPSec, VPN, SSH tunnel, etc.
- Supports ping to validate connection status

Protocol Selection: HTTP

- Clear text protocol
  - Easy to read
- No limit on request header size
- No dedicated ping

Protocol Selection: HTTP

- Does not directly support providing TLS termination information
  - Can be added by httpd using custom headers
  - Can be processed by Tomcat using the SSLValve
- Supports encryption via HTTPS

Protocol Selection: AJP vs HTTP

- If terminating TLS at httpd and you need TLS the information
  - Use AJP
- If you need to encrypt the httpd to Tomcat channel
  - Use HTTP
- If you need both
  - Use HTTP
  - It is usually easier to pass TLS info over HTTP than it is to encrypt AJP
- In you need neither
  - Pick the one you are familiar with – debugging will be easier

httpd Module Selection

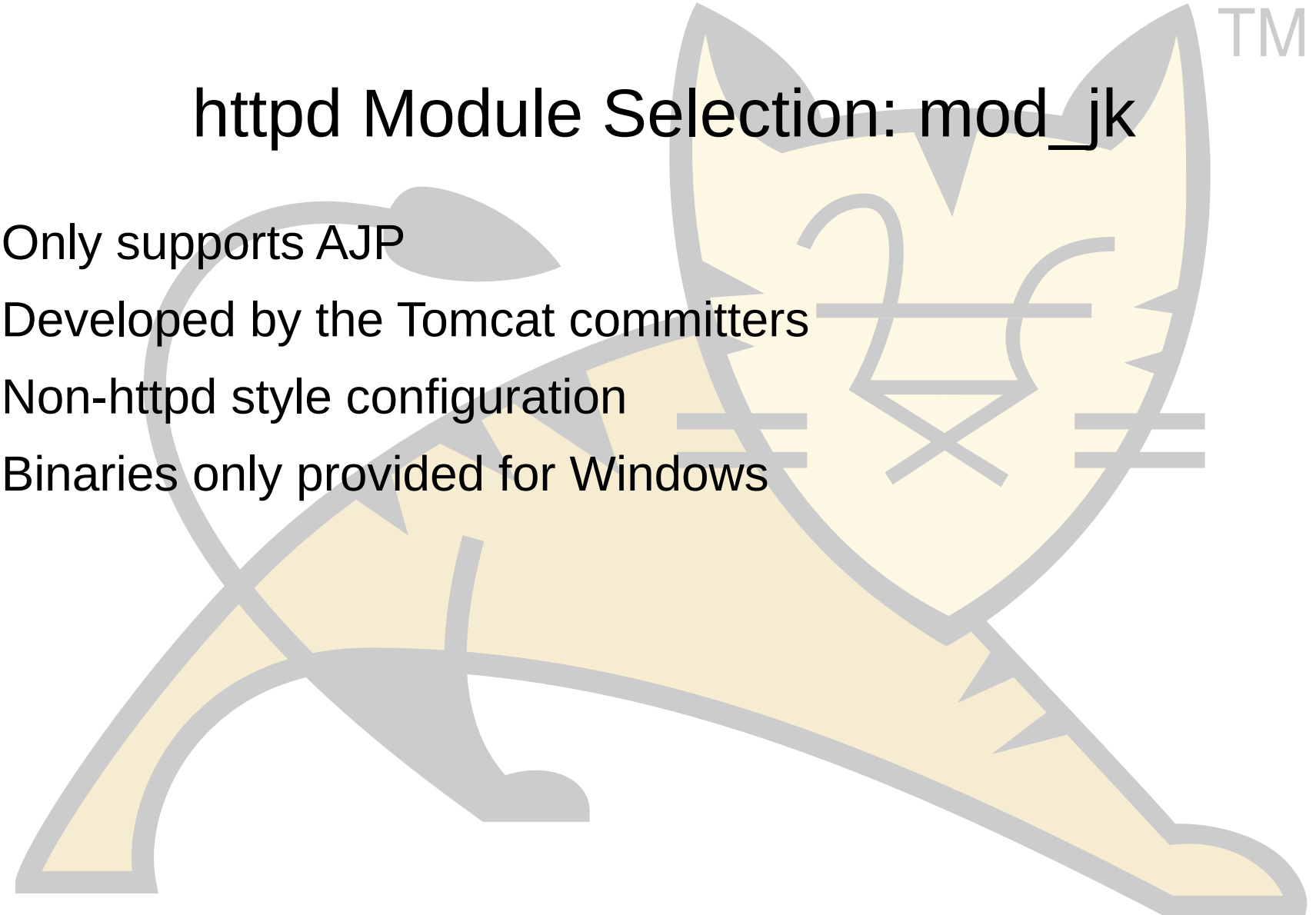
- Avoid
  - mod_jk2
  - mod_jserv
  - mod_webapp
  - anything else not explicitly mention below
- Consider
  - mod_jk
  - mod_proxy
  - (mod_rewrite)

httpd Module Selection: mod_rewrite

- You can replace most of httpd.conf with mod_rewrite directives
- That doesn’t mean that you should
- It is generally more efficient to use the dedicated directive
- There are times (complex load balancing rules) where I’ve used mod_rewrite
- mod_jk and mod_proxy can route based on environment variables
- Use mod_rewrite and/or mod_setenvif to determine the routing info
- Set the routing configuration with mod_jk / mod_proxy

httpd Module Selection: mod_jk

- Only supports AJP
- Developed by the Tomcat committers
- Non-httpd style configuration
- Binaries only provided for Windows



httpd Module Selection: mod_jk

- Doesn’t directly support URL re-writing
- Make sure you are using the latest documentation
- http://tomcat.apache.org/connectors-doc/
- The status worker can be used for monitoring and management

httpd Module Selection: mod_proxy

- Supports AJP and HTTP
- Included as standard with httpd
- Uses httpd style configuration
- Built-in support for URL re-writing (not all use cases)
- Binaries provided for most platforms

httpd Module Selection: mod_jk vs mod_proxy

- Mapping complexity no longer a differentiator
- Not on Windows and don’t want to have to compile the module
  - mod_proxy
- Already using one of these
  - Carry on. The costs of changing will probably out-weight the benefits
- If you have a free choice
  - Use mod_proxy, the configuration style will be more familiar

Tomcat Connector Selection

- BIO
  - Default in Tomcat 7
  - Removed from Tomcat 8.5 onwards
  - Java blocking I/O
- NIO
  - Default from Tomcat 8
  - Java non-blocking I/O
    - Next request, HTTP headers, TLS handshake

Tomcat Connector Selection

- NIO2
  - Introduced in Tomcat 8
  - Java non-blocking I/O
    - Next request, HTTP headers, TLS handshake
- APR/Native
  - JNI library built on APR non-blocking I/O
    - Next request, HTTP headers (8.5.x onwards)

Tomcat Connector Selection

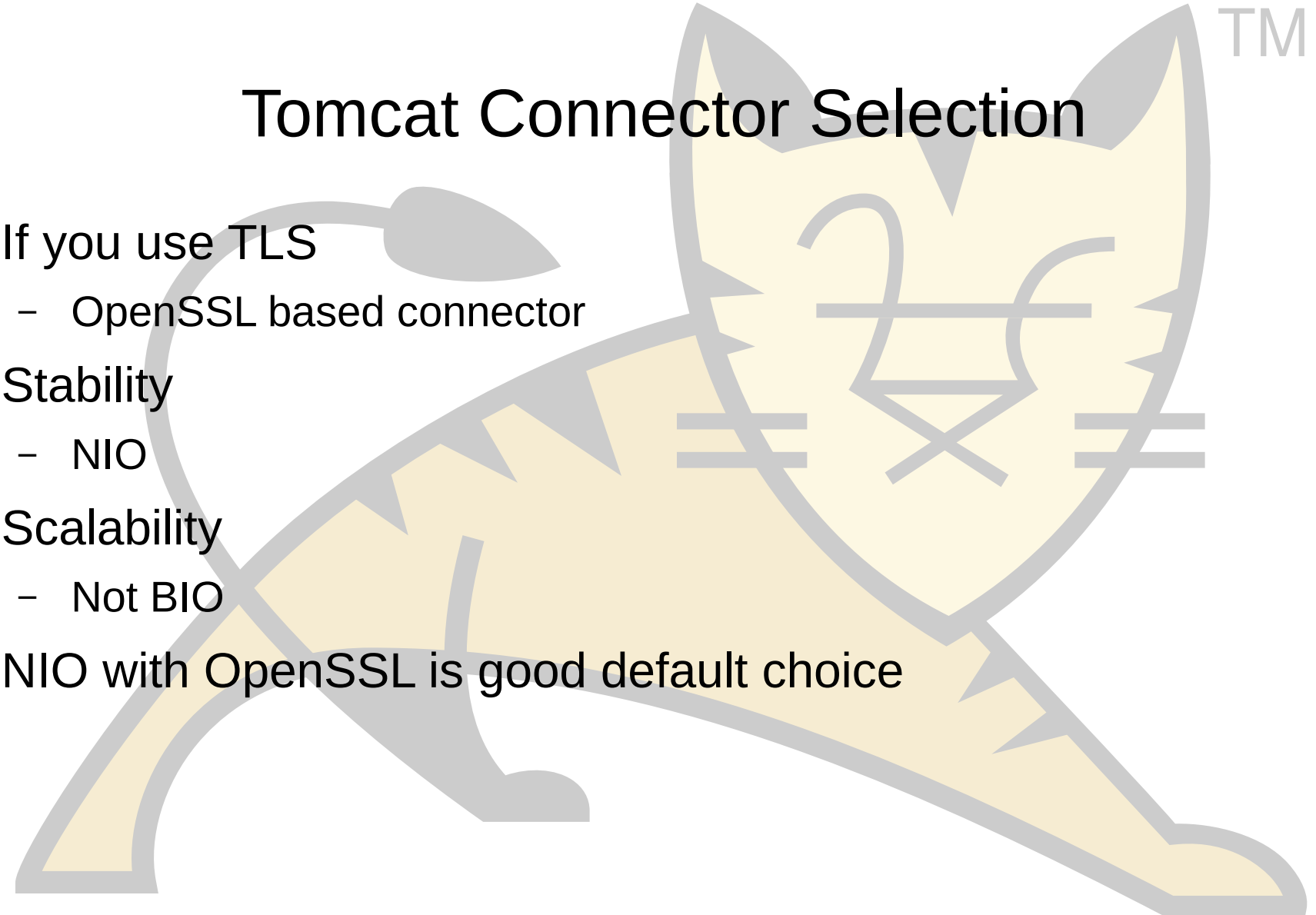
- All connectors block (or simulate blocking) during
  - Request body read
  - Response body write
- TLS
  - BIO, NIO & NIO2 uses JSSE
  - NIO & NIO2 can use OpenSSL from 8.5.x
  - APR/native uses OpenSSL
- OpenSSL is significantly faster

Tomcat Connector Selection

- Sendfile
  - NIO, NIO2 and APR/native support sendfile
- Comet
  - Removed in 8.5.x onwards
  - Supported by NIO, NIO2 and APR/native
- WebSocket
  - All connectors support WebSocket
  - httpd’s reverse proxy support doesn’t include HTTP upgrade
  - BIO fakes non-blocking support

Tomcat Connector Selection

- If you use TLS
  - OpenSSL based connector
- Stability
  - NIO
- Scalability
  - Not BIO
- NIO with OpenSSL is good default choice



Troubleshooting: Thread Exhaustion

- httpd prefork MPM
  - 1 thread per process
  - MaxRequestWorkers processes
  - Maximum of 1 * MaxRequestWorkers threads
- httpd worker MPM
  - ServerLimit processes
  - ThreadsPerChild threads for each process
  - Maximum of ServerLimit * ThreadsPerChild threads
- Thread == concurrent request

Troubleshooting: Thread Exhaustion

- Each httpd thread may create a connection to each Tomcat instance
- Therefore, 2 httpd instances each with 400 threads
  - Maximum of 800 connections to each Tomcat instance
  - The connections are NOT distributed between the Tomcat instances
  - Connections are persistent by default

Troubleshooting: Thread Exhaustion

- Connections may have low utilization
- BIO requires a thread per connection
- BIO connector may run out of threads even when Tomcat is almost idle

Troubleshooting: Thread Exhaustion: Solutions

- Use NIO connector as it is non-blocking between requests
- Don’t use persistent connections between httpd and Tomcat
- Ensure each Tomcat instance has >= threads than total httpd threads
- Configure timeouts
  - I have seen cases where httpd tried to use a timed out connection
- Use distance to create preferred groups
- Example: ASF Jira

Troubleshooting: Broken Links

- Easiest way to create a lot of hassle for yourself
  - ProxyPass /foo http://localhost:10180/bar
- Easiest way to avoid the hassle
  - ProxyPass /foo http://localhost:10180/foo
- Don’t change the context path

Troubleshooting: Broken Links

- Often marketing wants http://buzzword.com rather than http://buzzword.com/app
- Consider a simple redirect from / to /app
  - /app becomes visible to end users once they use the app
  - Much easier to implement and maintain
- Deploy your application as ROOT
  - Use ROOT##label if you need to add a version number or similar

Troubleshooting: Broken Links

- Redirects
  - Redirect to wrong path
- Cookie paths
  - Cookies are not returned by client
- Links
  - Created for wrong URL
- Custom headers (e.g. Spring MVC)

Troubleshooting: Broken Links: Solutions

- Fixing redirects
  - Don’t change the context path
  - ProxyPathReverse will fix some but not all HTTP headers
- Fixing cookie paths
  - Don’t change the context path
  - ProxyPassReverseCookiePath /bar /foo

Troubleshooting: Broken Links: Solutions

- Fixing links
  - Don’t change the context path
  - mod_sed, mod_substitute, mod_proxy_html
  - Fragile solution and a significant maintenance overhead

- Fixing custom headers
  - Don’t change the context path
  - mod_headers

Troubleshooting: Security Issues

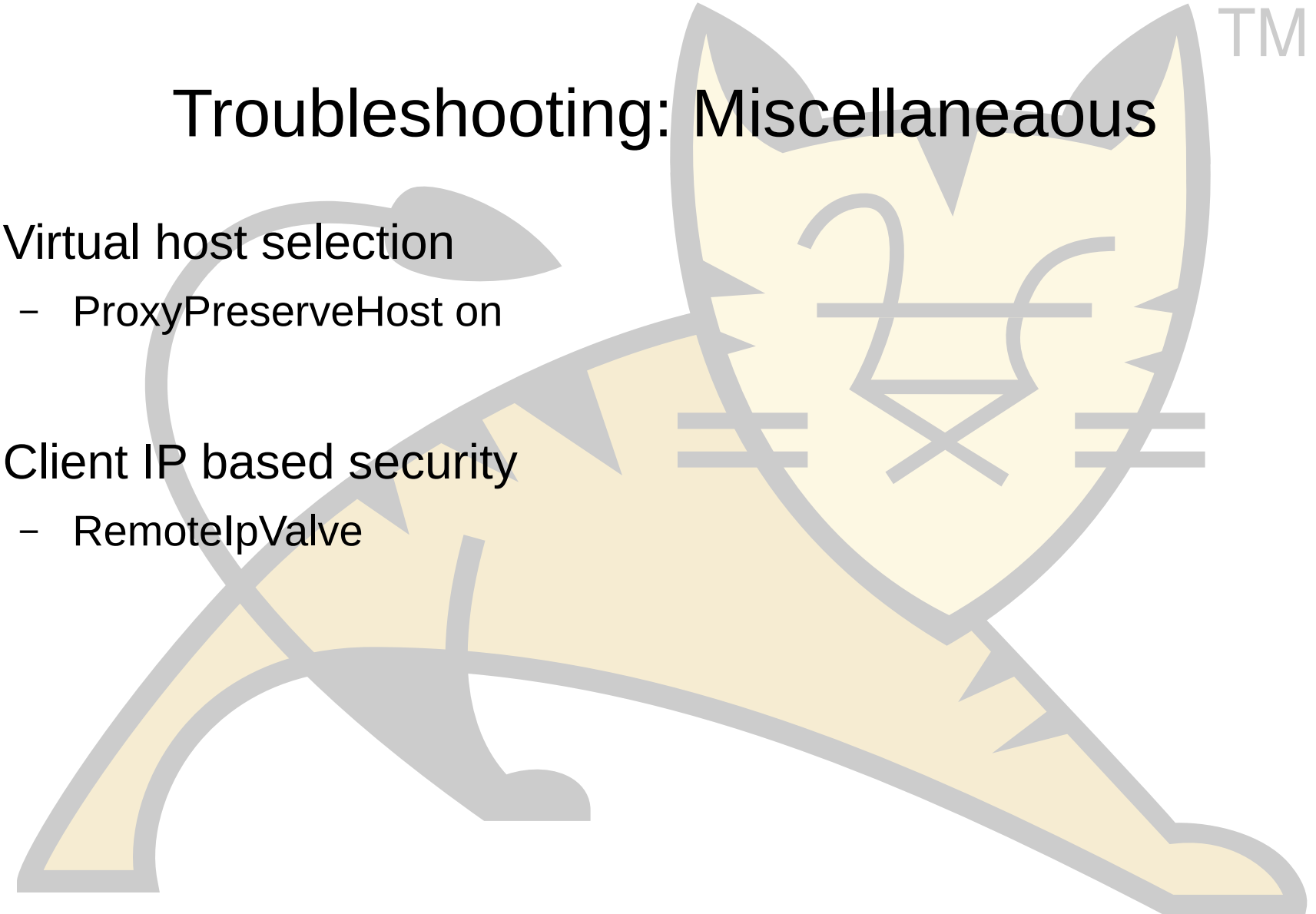
- Need to be careful when terminating HTTPS at httpd
- Tomcat needs to know if request was received over HTTPS
  - Sessions must not transition from HTTPS to HTTP
  - Cookies created over HTTPS must be marked as secure
- mod_jk and mod_proxy_ajp just handle this
- mod_proxy_http does not

Troubleshooting: Security Issues: Solutions

- Custom headers and the RemoteIpValve
- Two HTTP connectors
  - HTTP traffic proxied to connector with secure="false"
  - HTTPS traffic proxied to connector with secure="true"

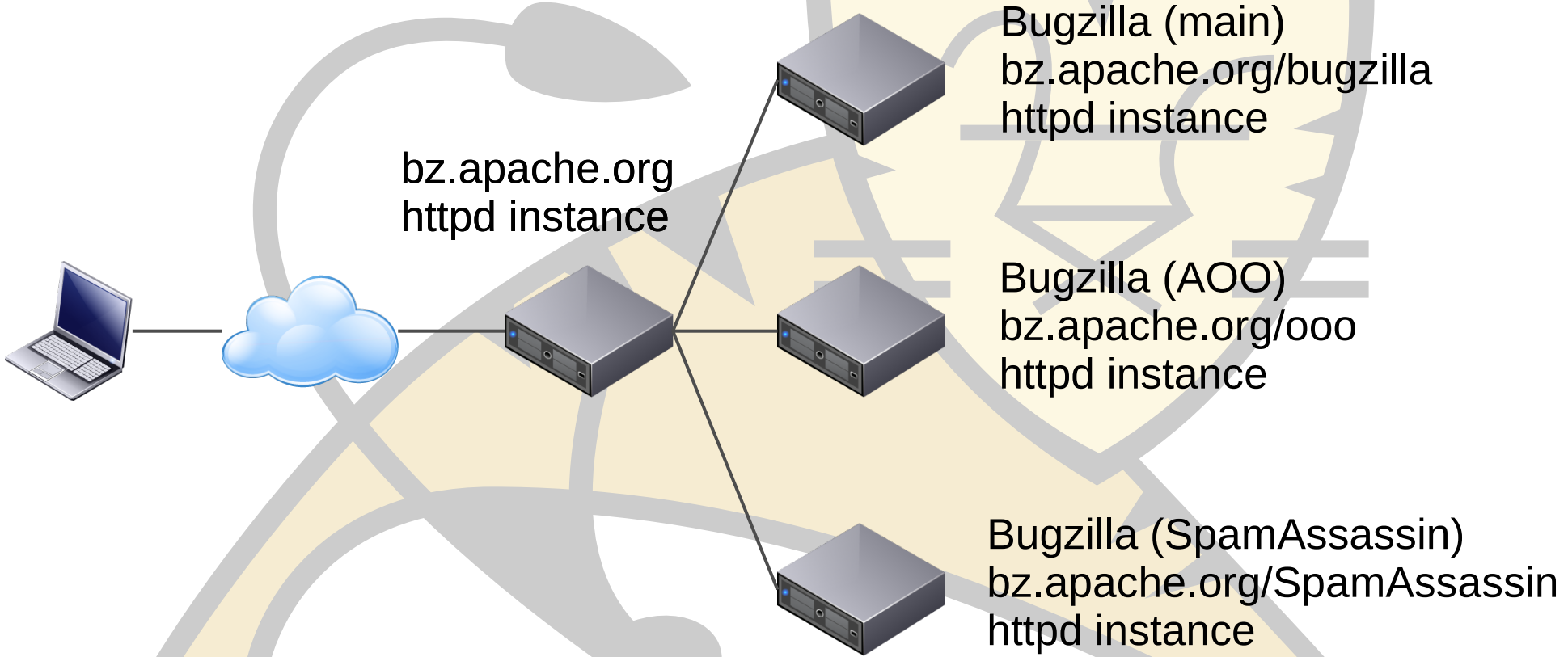
Troubleshooting: Miscellaneaous

- Virtual host selection
  - ProxyPreserveHost on
- Client IP based security
  - RemoteIpValve

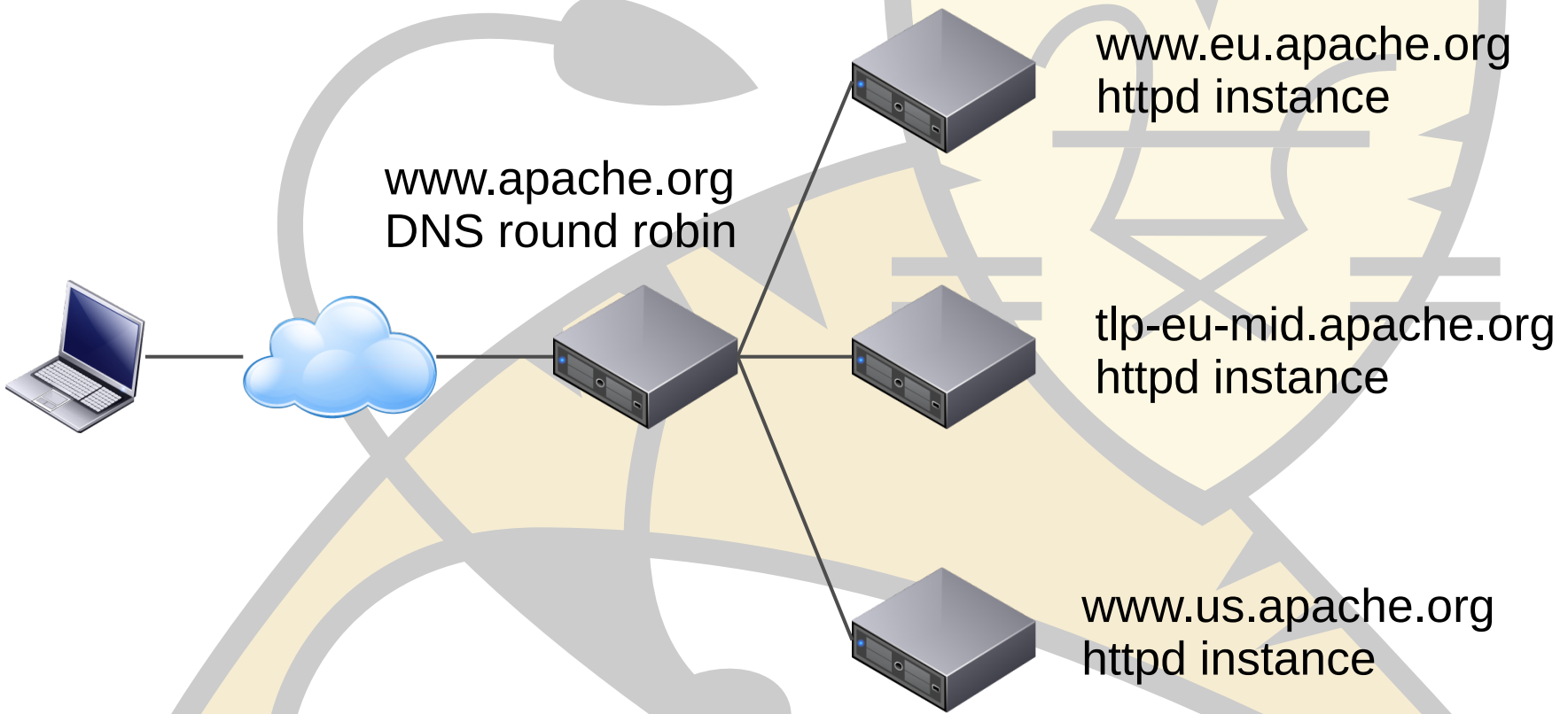


Load-balancing

Reverse Proxy



Load-balancing

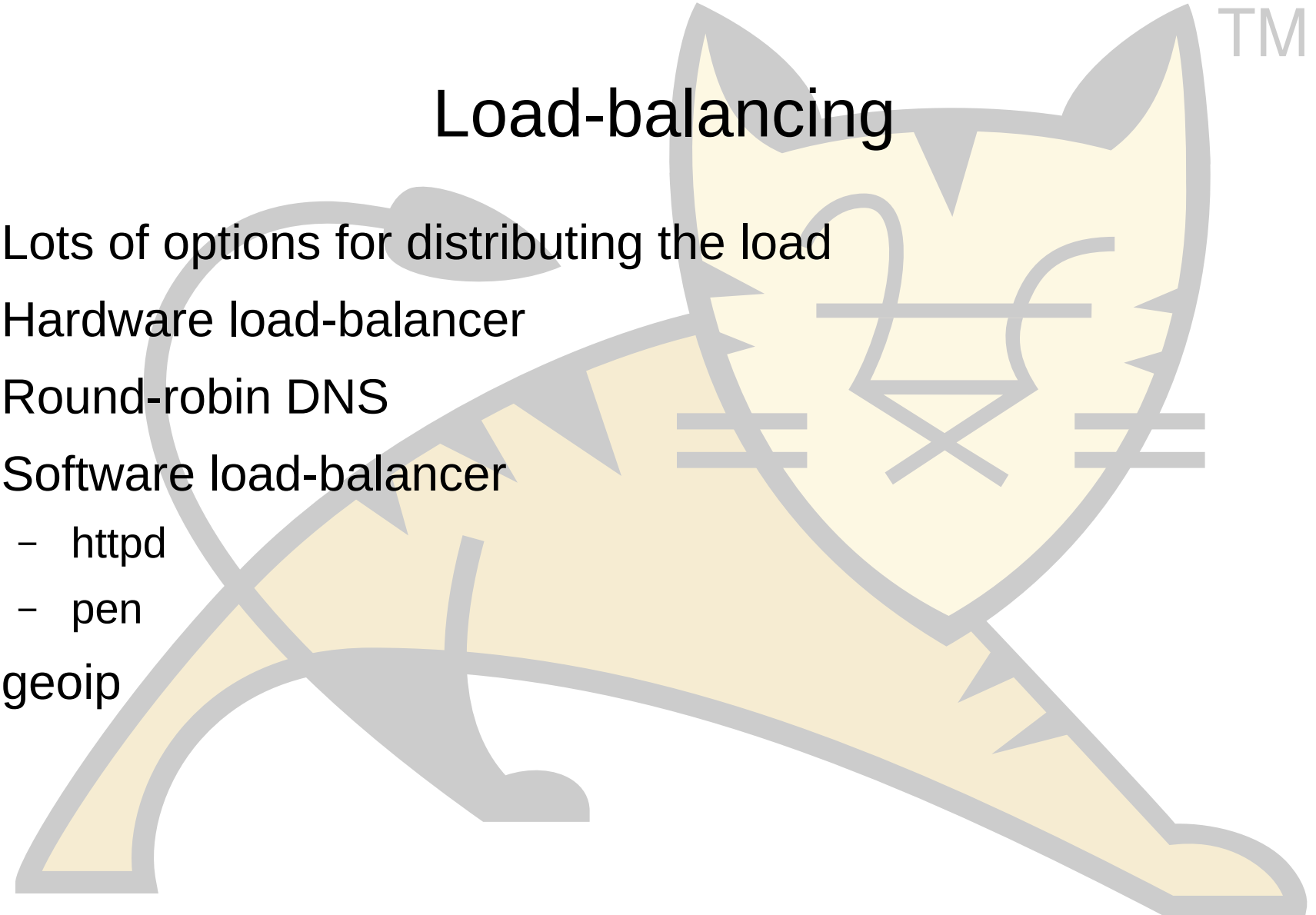


Load-balancing

- Looks like a single host to the clients
- Multiple hosts
- Each host is the same
- Each host is independent
  - No shared state between the hosts
  - May share common services (e.g. authentication, database)
- Node failure may be visible to users

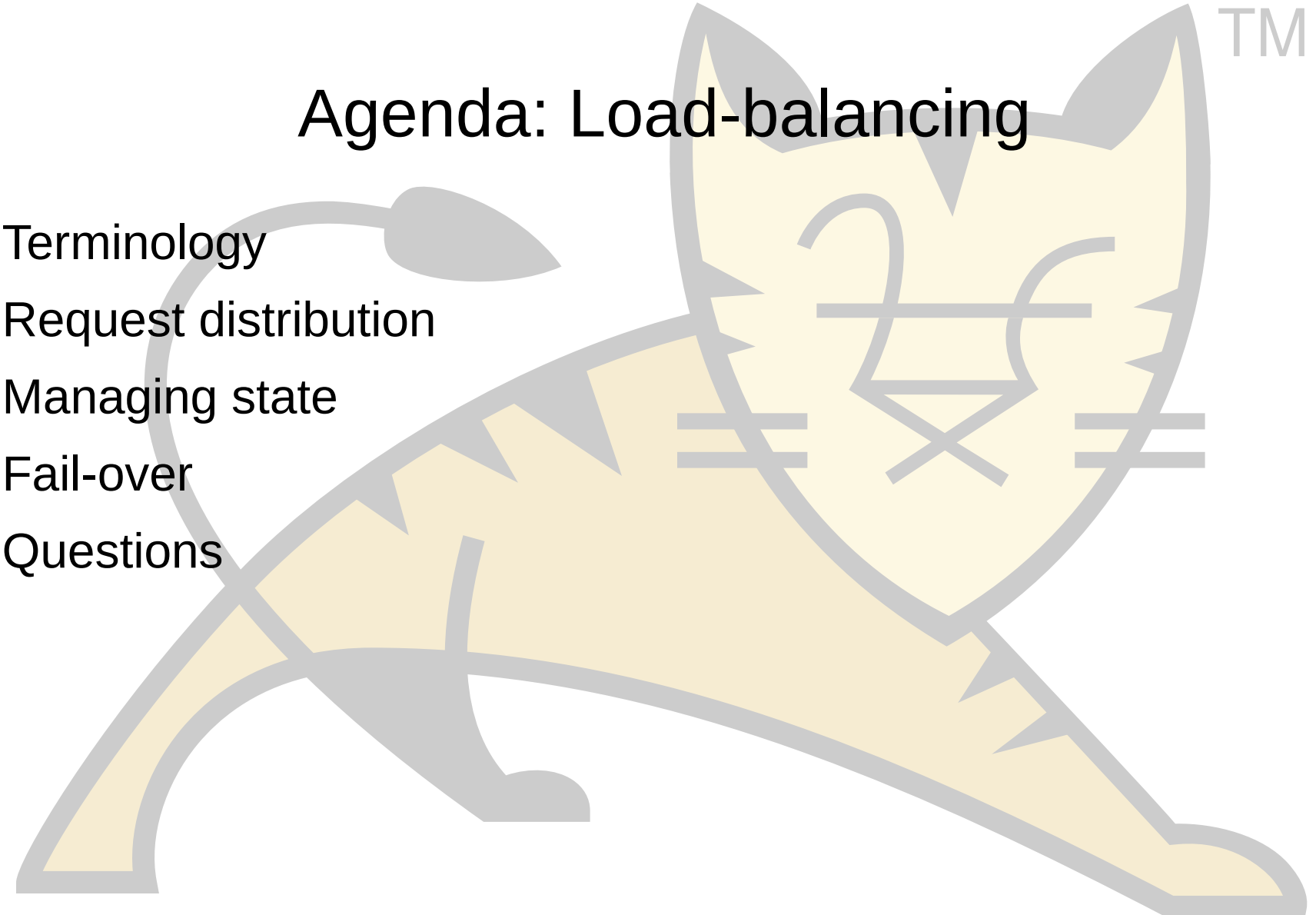
Load-balancing

- Lots of options for distributing the load
- Hardware load-balancer
- Round-robin DNS
- Software load-balancer
  - httpd
  - pen
- geoip



Agenda: Load-balancing

- Terminology
- Request distribution
- Managing state
- Fail-over
- Questions



Terminology

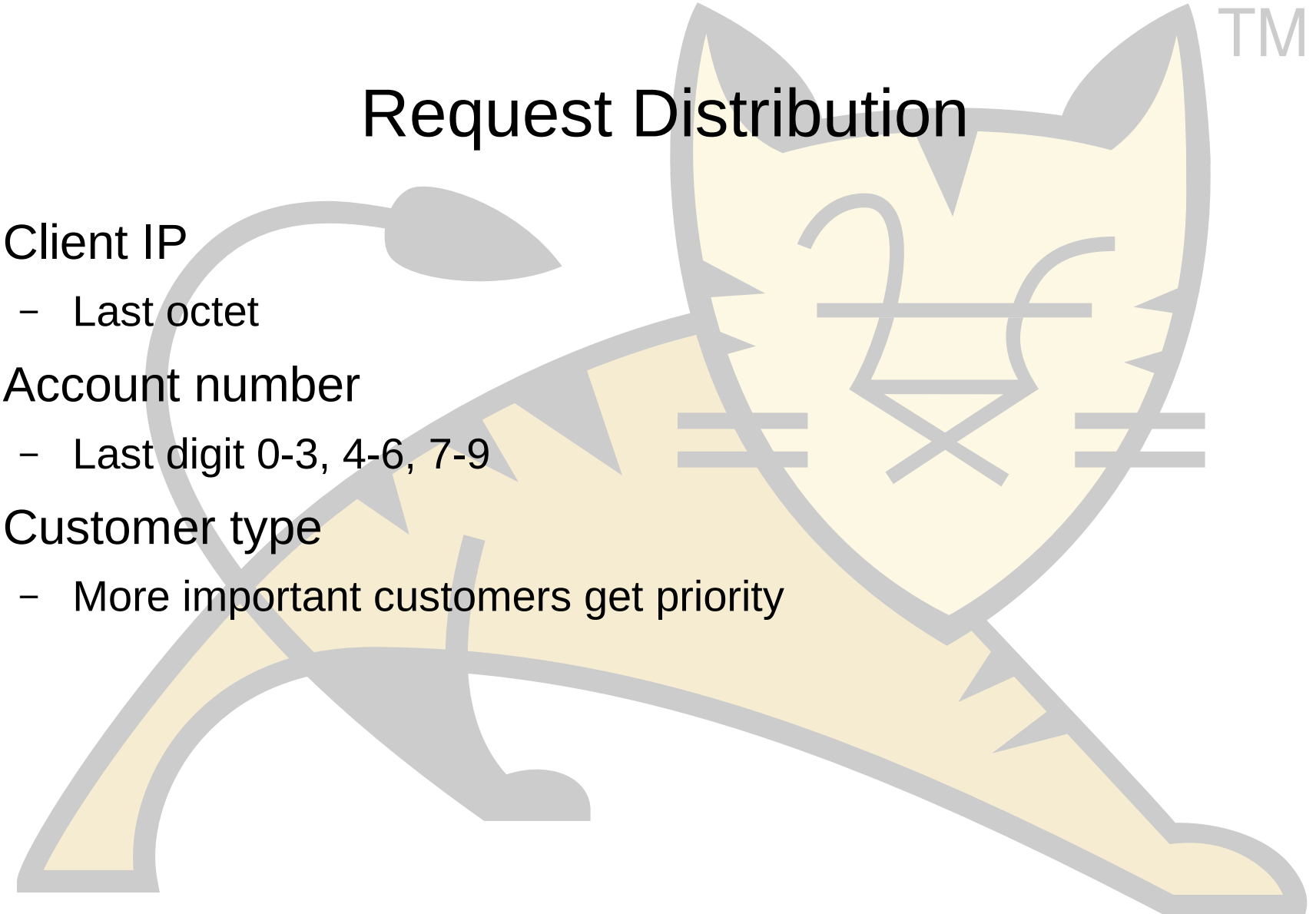
- Sticky sessions
- Without clustering, session is created only on node that handled request
- On next request, the load-balancer could send user to a different node where the session doesn’t exist
- Sticky sessions is a mechanism (there are several) that ensures the user returns to the node holding their session

Request Distribution

- Many ways to select node to handle request
- mod_proxy
  - Number of requests
  - Number of bytes returned
  - Number of current requests
- mod_jk
  - As mod_proxy plus
  - Number of sessions (estimate)

Request Distribution

- Client IP
  - Last octet
- Account number
  - Last digit 0-3, 4-6, 7-9
- Customer type
  - More important customers get priority



Managing State

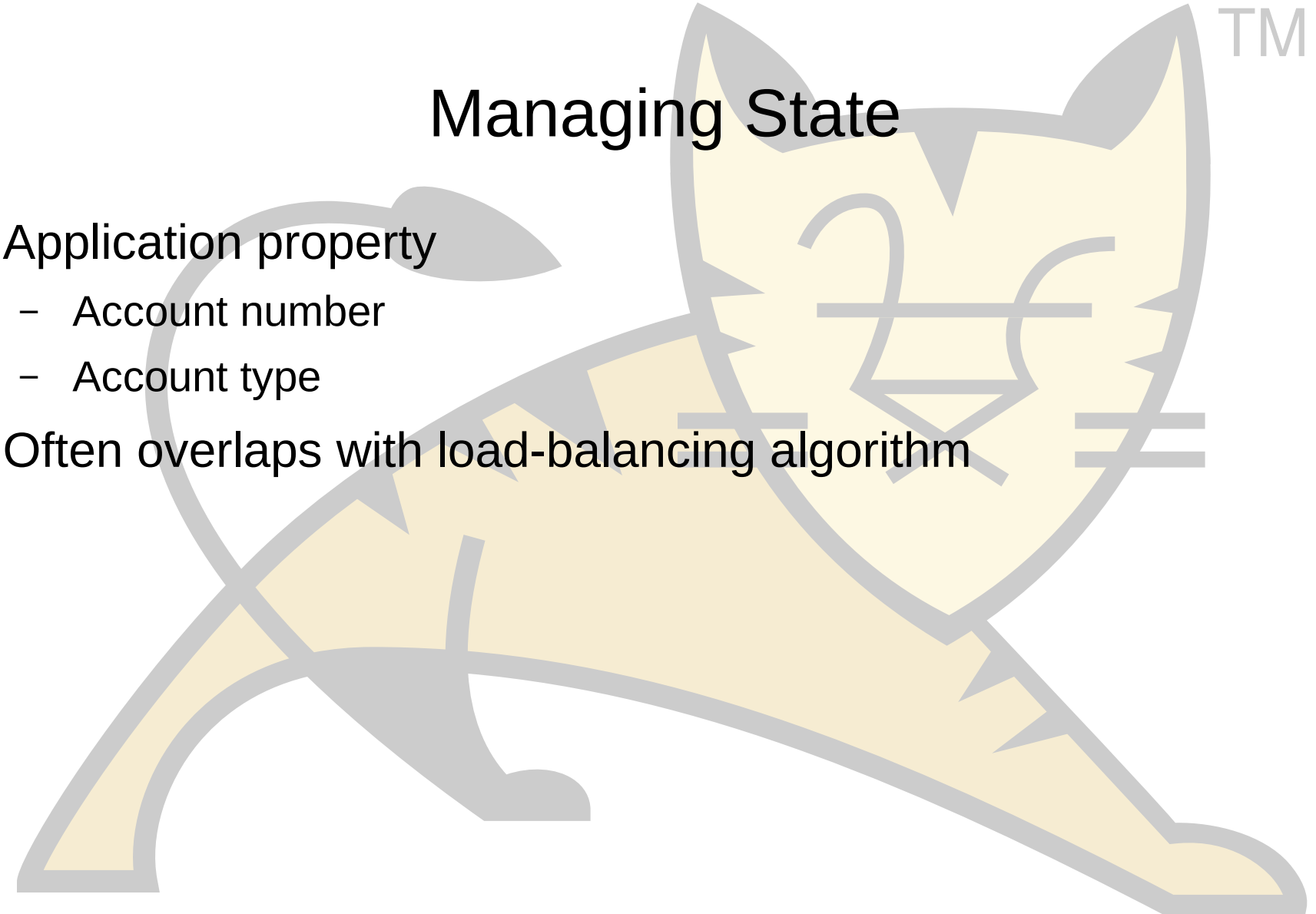
- Stateless applications are the simple solution
- Application state
  - State includes authentication
- Options
  - HTTP session
  - Database
  - Request parameters
- Load-balancing is impacted by HTTP state

Managing State

- Sticky sessions are used for HTTP State
- Session ID
  - Something in the session ID identifies the correct node
  - Users could change this
- Dedicated cookie
  - Users could change this
- Property of client such as IP
  - Beware of ISP that use forward proxies

Managing State

- Application property
  - Account number
  - Account type
- Often overlaps with load-balancing algorithm



Fail-over

- Load-balancer needs to know the state of the nodes
- Nodes need to taken off-line for maintenance
  - Known in advance
  - Several options
- Nodes will fail
  - Not (usually) predictable
  - Need to be able to detect dynamically
- What is the impact on users?

Fail-over: Maintenance

- More transparent to users means
  - More complex configuration
  - Process takes longer
- Need to drain node of users
  - How long can an HTTP session last?
  - At what point do you stop the node anyway?
- Can Tomcat’s parallel deployment feature help?

Fail-over: Unexpected

- Typically there is no separate management channel between Tomcat instances and load-balancer
  - There is with mod_cluster from RedHat
- Need to detect failed nodes so fail-over can happen as early as possible

Fail-over: Unexpected

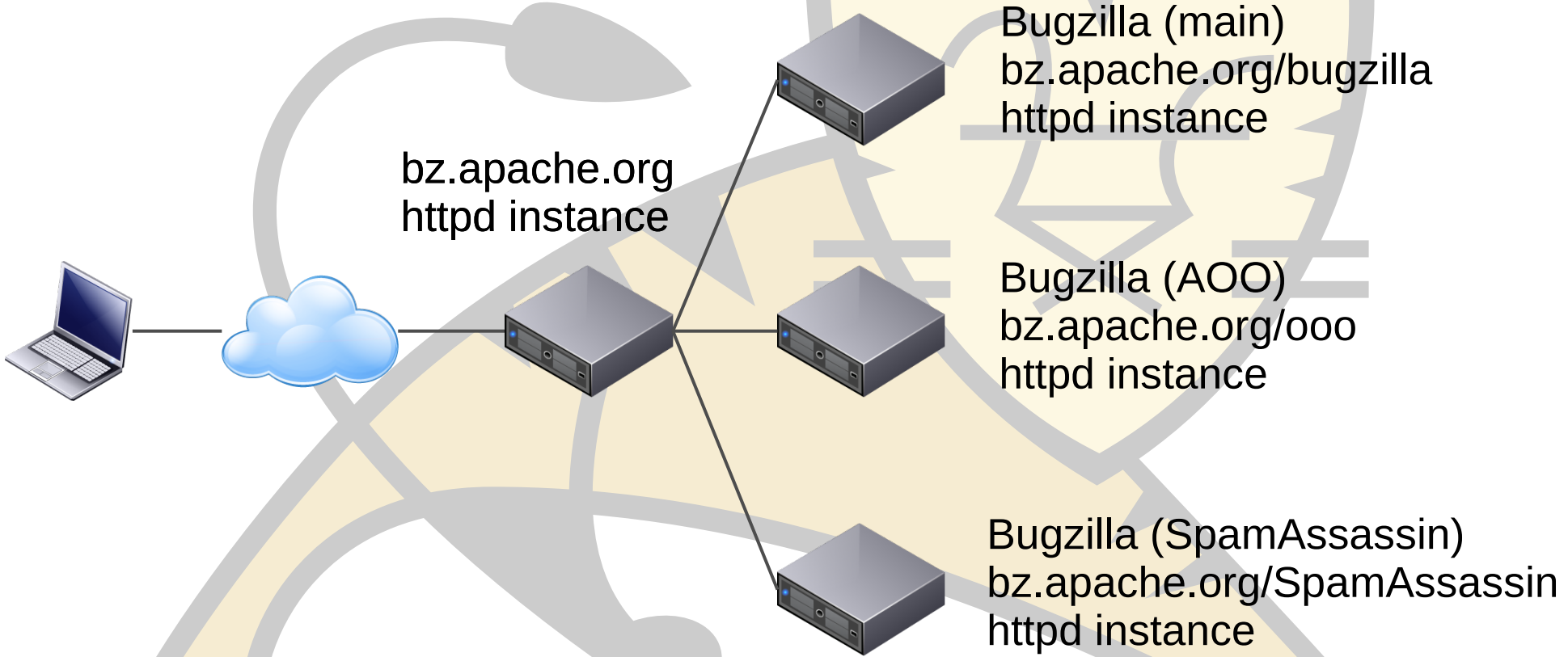
- Can use a ‘failed’ request to detect a failed node
- Is a 500 response because the server crashed or because of an application bug?
- Is a time-out because the server crashed or because it is just a long running request?
- Applications that can have long running requests take at least that long to detect failures.

Fail-over: Unexpected

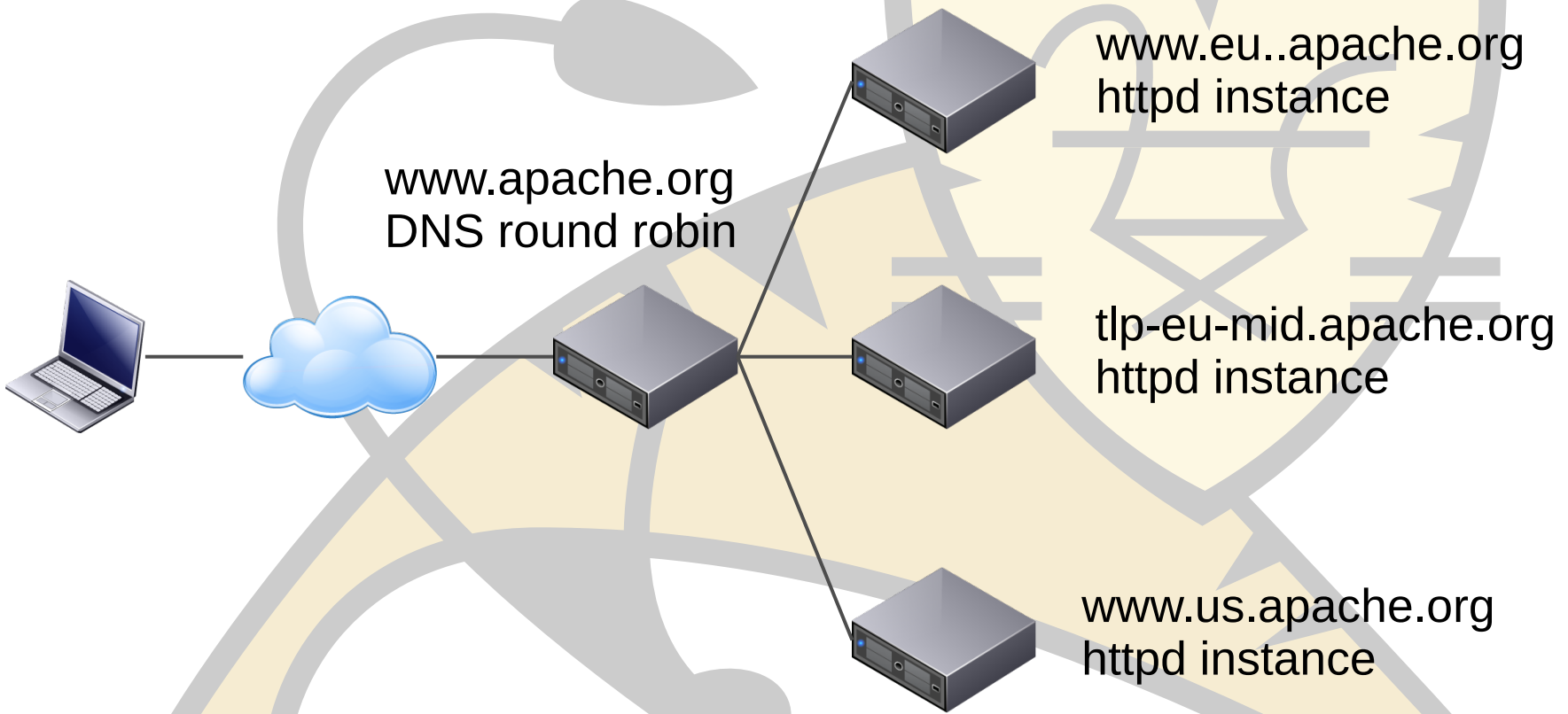
- Monitoring user initiated requests to detect node failure is fragile
- Load-balancer triggered request to known, working, ‘simple’ page
  - More reliable
  - Still an HTTP request with the associated overhead
- Protocol pings are even faster

Clustering

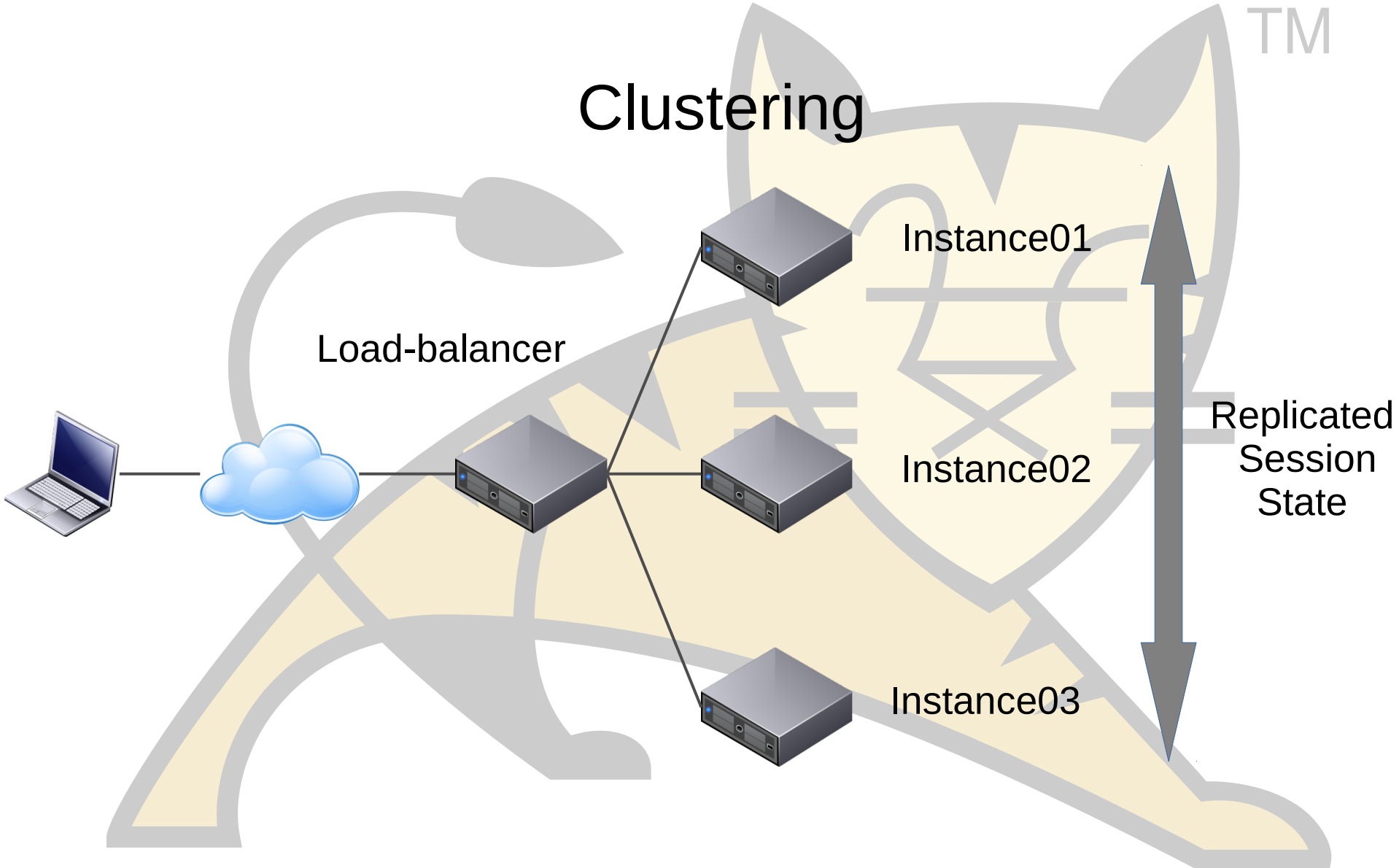
Reverse Proxy



Load-balancing

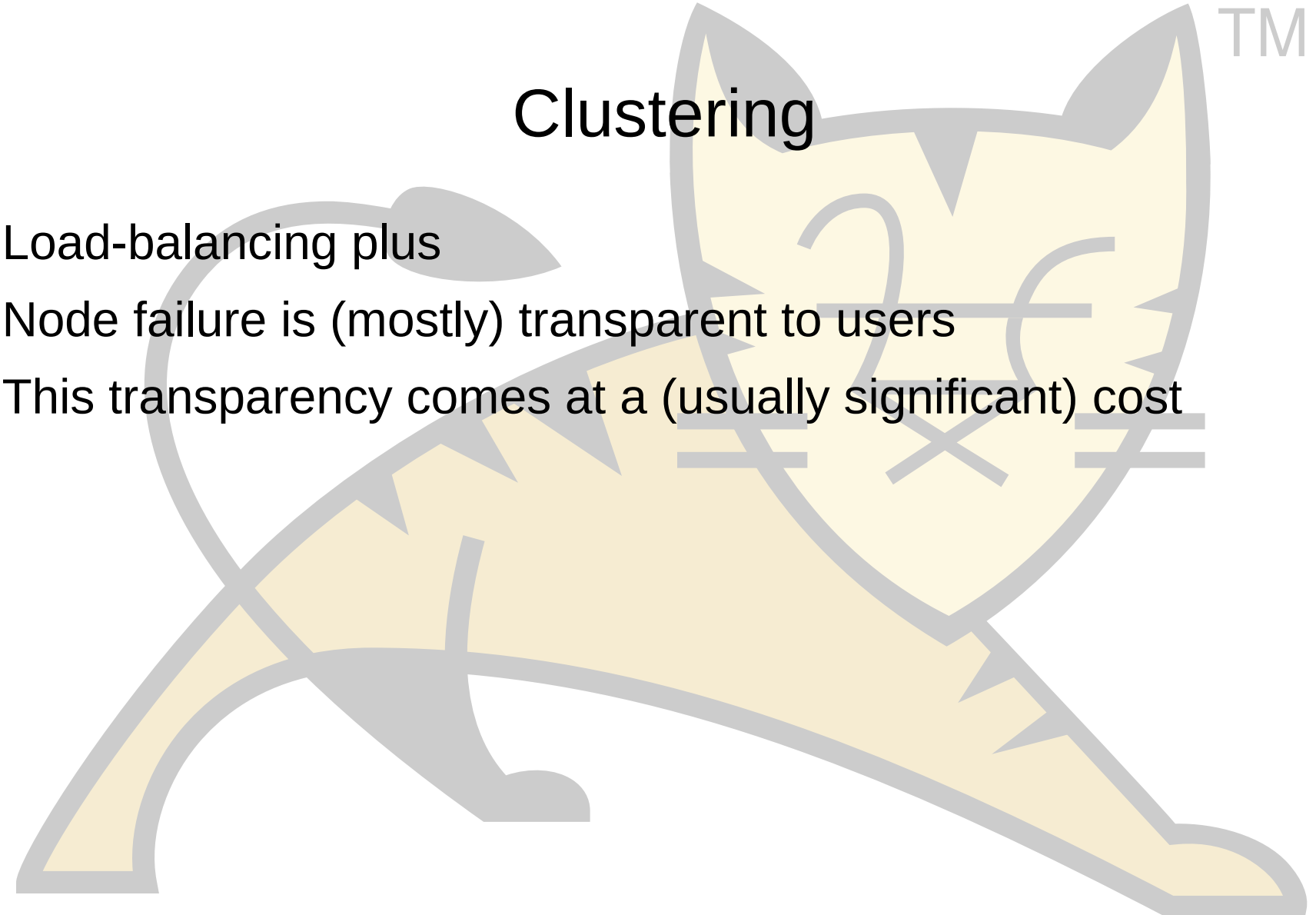


Clustering


Instance01
Load-balancer
Instance02
Replicated
Session
State
Instance03

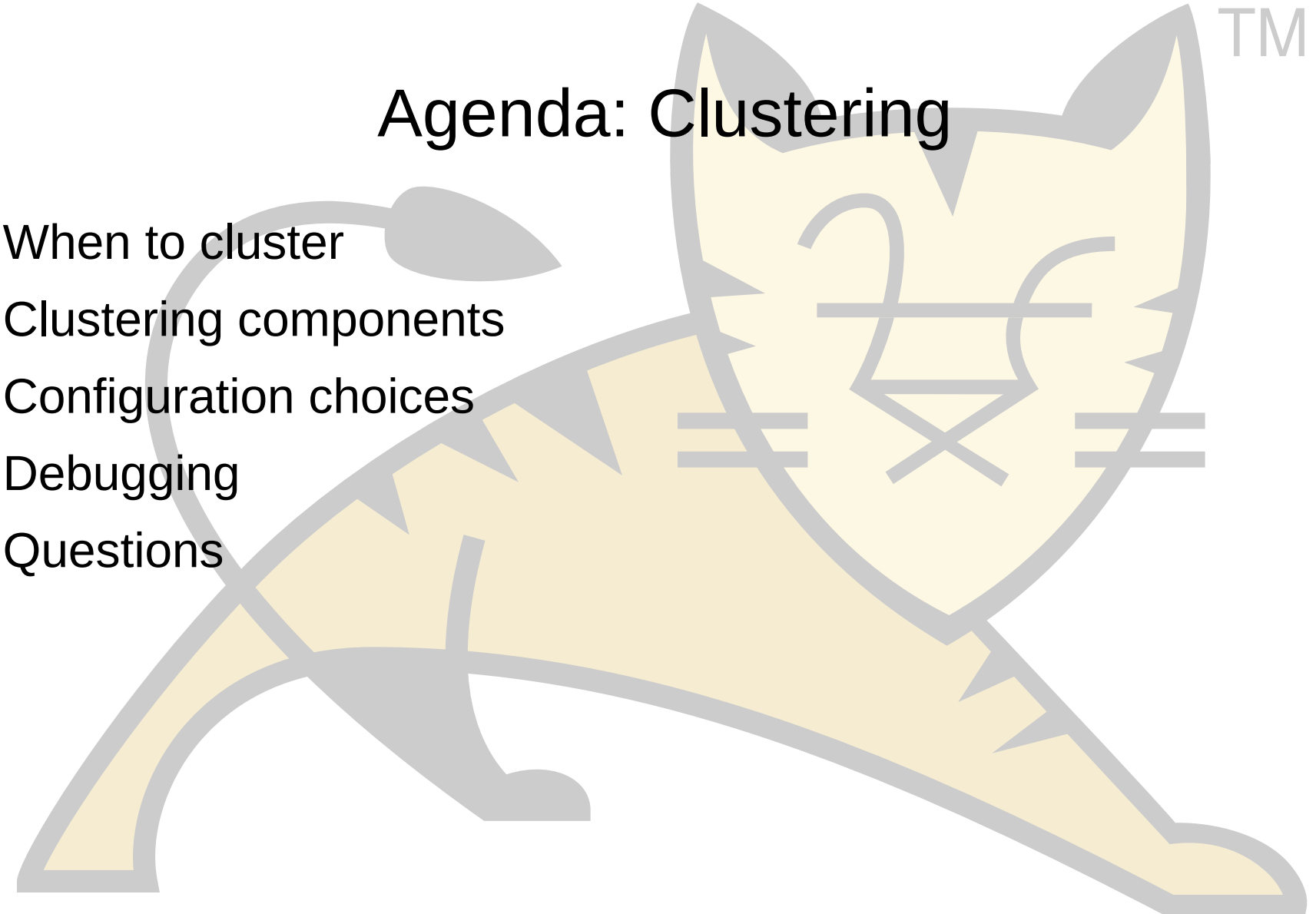
Clustering

- Load-balancing plus
- Node failure is (mostly) transparent to users
- This transparency comes at a (usually significant) cost



Agenda: Clustering

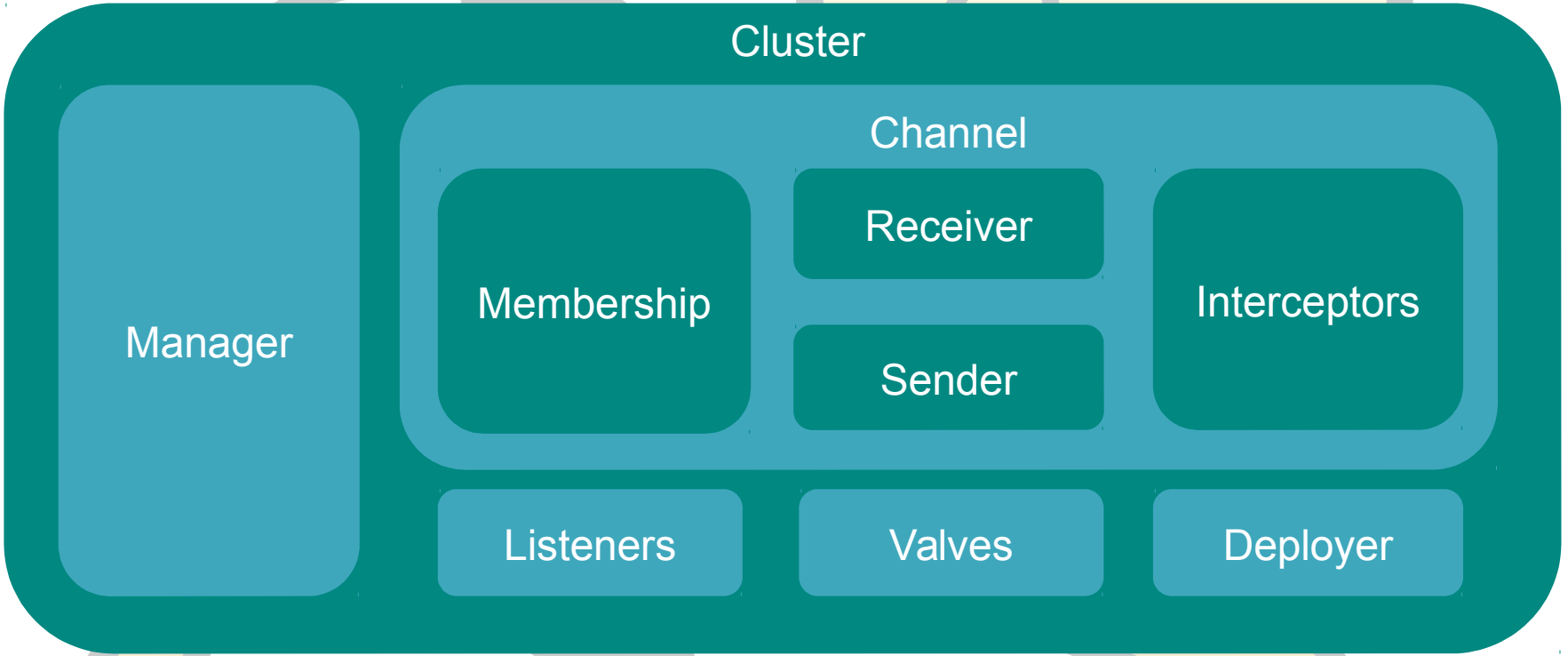
- When to cluster
- Clustering components
- Configuration choices
- Debugging
- Questions



When to cluster?

- Ideally, never
  - Adds configuration complexity
  - Requires additional processing
  - Debugging is lot harder
- What do you really need?
  - Load-balancing plus sticky sessions
  - If a node fails, sessions will be lost
- Clustering should be the last resort

Components


Cluster
Manager
Channel
Membership
Receiver
Sender
Interceptors
Listeners
Valves
Deployer

Components

- Cluster
  - Container for all cluster related configuration
  - May be placed within Engine or Host
- Manager
  - Controls how data is replicated between nodes
- Channel
  - Communication between cluster nodes

Components

- Membership
  - Tracks which nodes joining and leaving the cluster
- Sender
  - Sends cluster messages to other nodes
- Receiver
  - Receives messages from other nodes
- Interceptors
  - Valves for cluster messages

Components

- Listeners
  - Receive notifications of cluster messages
  - Managers support LifecycleListener
  - Standard session listeners remain available
- Valves
  - Inserted into the request processing pipeline
- Deployer
  - Cluster wide deployment of web applications

Manager Selection: Delta Manager

- Default
- Replicates every change to every node
  - Maximum reliability
- Network traffic proportional to the square of the number of nodes
  - Doesn’t scale to large numbers of nodes
- Fail-over can be to any node

Manager Selection: Backup Manager

- Sessions have a primary node and a backup node
  - Need to use sticky sessions
- Backup node selected on a round-robin basis from all other nodes
- There is NOT a single backup node
- Every node knows the primary node and backup node for every session
- Network traffic proportional to the number of nodes
- Failover is more complicated

Backup Manager Fail-over

Node A
Primary Sessions:
30*A
Backup sessions:
10*B', 10*C', 10*D'

Node B
Primary Sessions:
30*B
Backup sessions:
10*A', 10*C', 10*D'

Node C
Primary Sessions:
30*C
Backup sessions:
10*A', 10*B', 10*D'

Node D
Primary Sessions:
30*D
Backup sessions:
10*A', 10*B', 10*C'

Backup Manager Failover

Node A
Primary Sessions:
30*A
Backup sessions:
10*B', 10*C', 10*D'

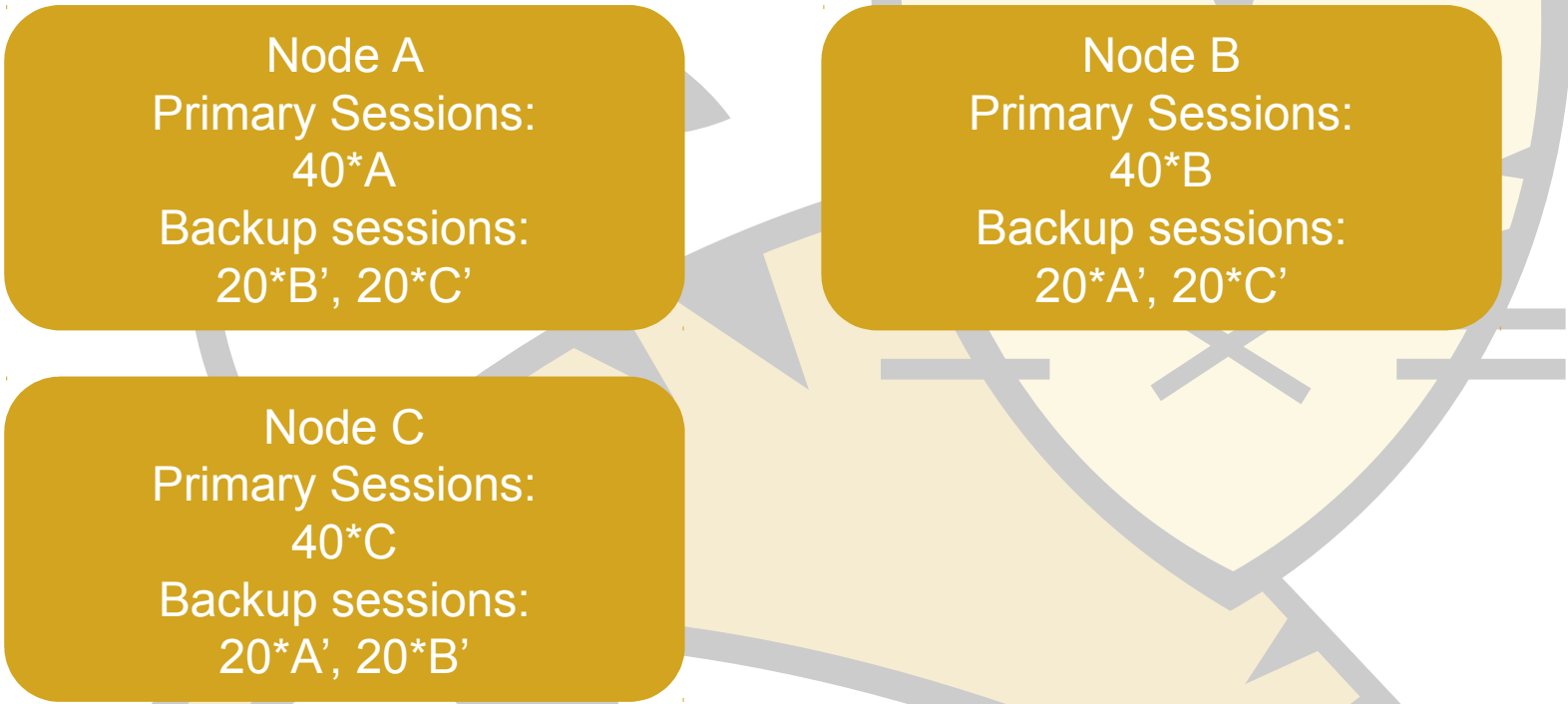
Node B
Primary Sessions:
30*B
Backup sessions:
10*A', 10*C', 10*D'

Node C
Primary Sessions:
30*C
Backup sessions:
10*A', 10*B', 10*D'

Backup Manager Fail-over

- Node D fails
- Sessions will be distributed to other nodes
  - As soon as node failure is detected
- Backup node becomes the primary
- A new backup node is selected
- Session is copied to new backup node

Backup Manager Fail-over



Backup Manager Fail-over

- Not quite that simple
- Load-balancer will redistribute sessions
- New primary chosen by load-balancer may not be the same as chosen by clustering
- More moving of primary (and possibly backup)
- End result is – on average – the same

Membership

- Multicast membership
  - Requires multicast to be enabled on the network
  - Can be difficult to debug problems
  - Scales more easily
- Static
  - Simple to debug
  - Adding nodes gets time consuming as cluster grows

Send Options

- Delta manager
  - channelSendOptions on Cluster
- Backup manager
  - mapSendOptions on Manager

- Synchronous or asynchronous

Send Options: Synchronous

- Request processing does not complete until session data has been sent
- What is meant by sent?
  - On the TCP stack
  - Received by the other node
  - Processed by the other node
- Next request to a different node will see updated sessions

Send Options: Asynchronous

- Request processing continues while session data is sent
- Next request to a different node may or may not see updated sessions

- Requires sticky sessions
- Fail-over may silently lose data

Debugging: Cluster Configuration

- Need to know
  - Session ID
  - Current route
  - Which node handled the request
- I use a simple JSP page that shows the above

Debugging: Cluster Configuration

- Sanity check
  - Is the route correct for the current node?
  - Is load-balancing happening as expected?
  - Is fail-over happening as expected?
- Keep in mind how reverse proxy / load-balancer handles failed nodes

Debugging: Applications

- Just like trying to debug any other application problem
  - But harder
- Can the issue be replicated in a non-clustered environment?
- Approach depends a lot on the application

Debugging: Applications

- Network / failover issues
  - Look at the access logs (need session IDs)
  - Look at error logs
  - May need to look at network traffic
- Application issues
  - Logging, logging and more logging
  - Need to be able to fine tune logging

Questions

Thank you